

GSM module HTTP GET & POST request using arduino microcontroller and SIM900

Published: June 27, 2025 at 2:00 AM Author: Richard Djarbeng 12 min read

This post covers the procedure to send a GET and POST request with AT commands using the SIM900 GSM/GPRS module and the Arduino UNO. It is useful for retrieving data from the Web and sending data to a web server for Internet of Things(IOT) or embedded systems projects. Also this should work with similar GSM modules that are operated by AT commands such as SIM800L, SIM808, SIM7000e etc. This tutorial uses 2G (GSM) connection.

I would be glad if it could be updated for 3G and 4G modules, it's a pity 🙄 we have to be using 2G networks for projects in 2025.

This tutorial assumes you are using an arduino uno but it should work for other arduino boards with minor changes. If you are using a raspberry pi microcontroller do **NOT** connect the raspberry pi directly to the SIM900. The SIM900 uses 5V logic level and the raspberry pi uses 3.3V logic level. For raspberry Pi users see the raspberry Pi section below.

Components:

- [Arduino Uno microcontroller](#) - for running the commands
 - Serial cable for arduino Uno - for connecting the arduino to your pc
 - Power supply for SIM900. Recommended is a 5V power supply that can provide 2A, but 9V 1A, or 12V 1A also works.
 - Registered SIM card with a data plan - for accessing the internet
-
- SIM900 GSM/GPRS module

Connection diagram

If you need help connecting the SIM900 to the arduino you can check out this tutorial: [Guide to SIM900 GSM GPRS Shield with Arduino](#)

Here's the connection diagram anyway.

From my experience, the second connection of the ground cable to the pin on the SIM900 coloured yellow away from the RX,TX, did not make a difference in the behaviour.

The following sections provide a breakdown of the code by sections but you can skip to the full code below if you prefer.

Setup

To use the SIM900 GSM module with an Arduino Uno, you first need to import the SoftwareSerial library and define a SoftwareSerial object for communication with the module. Here's an example:

```
#include <SoftwareSerial.h>

// SoftwareSerial object for communication with SIM900
SoftwareSerial SIM900(2, 3); // RX, TX
```

In this example, we set the RX pin to digital pin 2 and the TX pin to digital pin 3. You can adjust these pin assignments based on your specific setup.

Next, there are two utility functions provided in the code that simplify sending commands to the SIM900 module and displaying the response. However, if you prefer, you can directly send data to the module using `SIM900.println("command");`, replacing "command" with the AT command you want to send.

The utility functions are as follows

```
void sendCommand(const char* command) {
    SIM900.println(command);
    ShowSerialData();
}

void ShowSerialData() {
    Serial.println("Show serial data:");
    while (SIM900.available()) {
        char c = SIM900.read();
        Serial.write(c);
    }
    Serial.println("");
    delay(1000);
}
```

The `sendCommand` function takes a command as input and sends it to the SIM900 module using `SIM900.println(command)`. After sending the command, it calls the `ShowSerialData` function to display the response received from the module.

The `ShowSerialData` function reads the response from the SIM900 module using `SIM900.available()` and `SIM900.read()`. It then writes each character of the response to the Serial monitor using `Serial.write(c)`. Finally, it adds a delay of 1 second (`delay(1000)`) to allow for easier reading of the displayed data.

Please note that these utility functions are provided to assist in displaying the results of commands in a structured manner. If you prefer, you can directly send commands to the SIM900 module using `SIM900.println("command");`. The choice between using the utility functions or direct command sending depends on your preference and the level of detail you want in displaying the response.

Working with the SIM900 - Attention is all you need 😊

The SIM900 GSM module communicates with AT commands, where **AT** stands for attention. To perform any action with the GSM module such as sending the GET and POST request you will send a series of AT-commands that look like this mostly:

```
AT+command  
AT+command=value1, value2
```

Most of these commands return a value indicating if the request was successful or not. Most successful commands end with OK but when a command is not successful the module usually returns 'ERROR'

The simplest AT-command is simply 'AT' which is used to check if the module is connected to the arduino. It returns 'OK' when successful, if not successful please check the connection to the SIM900 and try again because without the 'AT' command returning OK all other commands will most likely not work.

Connecting to the internet

GET request - Retrieving data from the internet

This code snippet demonstrates how to perform a GET request using the GPRS connection of the SIM900 GSM module. It utilizes various AT commands to establish the connection and send the request.

```
void sendGetRequest(){

    //Check if the module is responsive, expected value OK
    sendCommand("AT");
    //close or turn off network connection in case it was left open, expected value OK
    sendCommand("AT+CIPSHUT");
    // close GPRS context bearer in case it was left open, expected value OK
    sendCommand("AT+SAPBR=0,1");
    // open GPRS context establish GPRS connection
    sendCommand("AT+SAPBR=3,1,\"Contype\",\"GPRS\"");
    //Set the Access Point Name (APN) for the network provider
    //change this apn value for your SIM card
    sendCommand("AT+SAPBR=3,1,\"APN\",\"internet.mtn\"");
    //open GPRS context bearer
    sendCommand("AT+SAPBR=1,1");
    //initiate HTTP request
    sendCommand("AT+HTTPINIT");
    //set parameters for http session, HTTP context identifier
    sendCommand("AT+HTTTPARA=\"CID\",1");
    //Change the URL from google.com to the server you want to reach
    sendCommand("AT+HTTTPARA=\"URL\",\"http://example.com/\"");
    //Initiate the HTTP GET request, send http request to specified URL
    sendCommand("AT+HTTPACTION=0");
    // Wait for the response (adjust the delay as needed)
    delay(9000);
    // Read the HTTP response, normally contains status code 200 if successful
    sendCommand("AT+HTTPREAD");
    //Terminate the HTTP service
    sendCommand("AT+HTTPTERM");
    //close or turn off network connection
    sendCommand("AT+CIPSHUT");
    // close GPRS context bearer
    sendCommand("AT+SAPBR=0,1");

}
```

Explanation of AT commands for GET request:

The backslash " in the arduino code is used as an escape character so that the quotes " used in the command are not mistaken for the code for the Arduino

1. **AT**: This command checks the responsiveness of the SIM900 module. Expected response: "OK" if the module is responsive.
2. **AT+SAPBR=3,1,"Contype","GPRS"**: Sets the connection type to GPRS. Expected response: "OK" if the command is successful.
3. **AT+SAPBR=3,1,"APN","internet.mtn"**: Sets the Access Point Name (APN) for the GPRS connection. Please change the "internet.mtn" to the appropriate APN for your network provider. Expected response: "OK" if the command is successful.
4. **AT+SAPBR=1,1**: Opens the GPRS context to establish the connection. Expected response: "OK" if the connection is successfully established.

*Note: This command **AT+SAPBR=1,1** may fail when running it a second time and show "ERROR" but the GET request can still work.*

5. **AT+HTTPINIT**: Initializes the HTTP service on the SIM900 module. Expected response: "OK" if the initialization is successful.
6. **AT+HTTPPARA="CID",1**: Sets the HTTP context identifier to 1. Expected response: "OK" if the parameter is set successfully.
7. **AT+HTTPPARA="URL","<http://example.com/>"**: Sets the URL for the server. Replace "<http://google.com/>" with the actual URL of the server you want to send the GET request to. Expected response: "OK" if the URL is set successfully.
8. **AT+HTTPACTION=0**: Initiates the HTTP GET request. Expected response: "OK" followed by the HTTP response code (e.g., "+HTTPACTION:0,200") if the request is successful.
9. **delay(9000)**: Waits for the response from the server. Adjust the delay time as per your requirements.
10. **AT+HTTPREAD**: Reads the HTTP response from the server. Expected response: The response data from the server.
11. **AT+HTTPTERM**: Terminates the HTTP service on the SIM900 module. Expected response: "OK" if the termination is successful.

Please note that the APN and URL for the server are provided as hardcoded values in this code snippet. It is recommended to make them variables so that users can easily modify them without having to locate them in the code.

It is worth mentioning that the get request is not only used for retrieving data from the internet it can be used to send data in the following format `http://example.com?value1=2?`

`value2=13` Where value1 and value2 are the parameters being sent as 2 and 13.

POST request - Sending data to the internet

```
void testPostRequest(String jsonToSend){
    //Example format of JSON:
    // String jsonToSend="{\"uploadedAt\": \"2023-06-26T20:18:22.826Z\", \"data\": [{\"unit\": \"C\", \"reading\": 31}]}";

    sendCommand("AT");
    ShowSerialData();
    sendCommand("AT+CIPSHUT");
    ShowSerialData();
    delay(500);
    sendCommand("AT+SAPBR=0,1");
    delay(2000);

    ShowSerialData();
    sendCommand("AT+SAPBR=3,1,\"Contype\",\"GPRS\"");
    ShowSerialData();
    sendCommand("AT+SAPBR=3,1,\"APN\",\"internet.apn\"");
    ShowSerialData();
    sendCommand("AT+SAPBR=1,1");
    delay(2000);
    ShowSerialData();
    sendCommand("AT+HTTPINIT");
    delay(1000);
    ShowSerialData();
    sendCommand("AT+HTTPPARA=\"CID\",1");
    ShowSerialData();
    sendCommand("AT+HTTPPARA=\"URL\",\"http://example.com/data\"");
    ShowSerialData();
    sendCommand("AT+HTTPPARA=\"CONTENT\",\"application/json\"");
    ShowSerialData();
    sendCommand(("AT+HTTPDATA=" + String(jsonToSend.length()) + ",20000").c_str());
    delay(6000);
    SIM900.println(jsonToSend);
    delay(16000);
    ShowSerialData();
    sendCommand("AT+HTTPACTION=1");
    delay(20000);
    ShowSerialData();
    sendCommand("AT+HTTPREAD");
    ShowSerialData();
    sendCommand("AT+HTTPTERM");
    ShowSerialData();
    sendCommand("AT+CIPSHUT");
    ShowSerialData();
}
```

Explanation of AT commands for POST request:

The AT commands for the POST request are similar to the get request above except on these lines.

1. **AT+HTTPPARA="CONTENT","application/json"**: Indicates that the data will be in JSON format. Expected value: OK
2. **AT+HTTPDATA=lengthOfJson,20000**: Prepares the module to receive the JSON data. First parameter is the length of the JSON to be sent and the second parameter is the time in milliseconds to wait for the data to be received, in this case; `20000` = 20 seconds. The module normally responds 'DOWNLOAD' waiting for the JSON input. The next command is actually not an 'AT' command but just the JSON input with the given length of characters specified.
3. **jsonToSend** : Send just the JSON data. Expected response: OK If the number of characters are lower or higher than expected the command will fail with 'ERROR'.
4. **AT+HTTPACTION=1**: Initiates the HTTP POST request. Expected response: "OK", number of characters in HTTP response followed by the HTTP response code (e.g. "+HTTPACTION:42,201")
5. **AT+HTTPREAD**: Reads the HTTP response from the server. Expected response: The response data from the server, usually contains the json data you just transmitted with some additional details or error information in case the request returns anything other than 200 or 201 status code.
6. The rest of the commands are the same as the GET request code, where we close the connection and reset the bearer

```
AT+HTTPPARA="CONTENT","application/json"
AT+HTTPDATA=lengthOfJson,20000
AT+HTTPACTION=1
```

Note: The length of the characters in the JSON to send for the command `AT+HTTPDATA=` must be exact or the command will fail with 'ERROR'. Example: `AT+HTTPDATA=42,20000` this indicates that the JSON data is 42 characters in length. When successful the `AT+HTTPDATA=` command returns 'DOWNLOAD OK'.

Testing AT commands on the serial monitor

You can run the file [testSerial.ino](#) to test the serial commands one by one. Enter commands in the serial monitor and send them to the GSM modules and view the response.

Full arduino code

Find the full code here:

- [GET request code for arduino](#)
- [POST request code for arduino](#)
- [Test AT commands using serial monitor for arduino](#)

Note that you will have to replace the URL based on the server you are sending the data to and the APN depending on the SIM provider you are using. The delays in the code can be adjusted accordingly. Generally the commands such as entering the JSON data and sending the request to the server requires more delays since their time to receive a response can change based on certain conditions. This is to prevent querying for a response too early.

Raspberry Pi- Note for raspberry pi users and SIM900

It is not recommended to connect the SIM900 board directly to the RX and TX (UART) pins of the raspberry Pi, because the SIM900 uses a 5V logic level and the raspberry Pi uses 3.3V. Connecting them directly could damage the Pi. You may have to use a voltage level shifter. Unfortunately, I don't know any recommended level shifters, if you do let me know; submit a PR or open an issue to get my attention.

Alternative GSM module for Raspberry Pi

The SIM7000e e-paper NB-IOT/GPRS HAT for the raspberry pi looks like a good alternative for Raspberry Pi users. It fits unto the raspberry pi and does not require cable connections. The GET and POST request commands were tested on that module and the commands were verified to work. You can also debug with Minicom program in place of the arduino serial monitor to send AT-commands individually to the module.

Other projects related to GSM GET and POST

- [TinyGSM](#) - arduino library for working with GSM modules
- [SIM900 Basic request library](#)

Appendix- About

This page was created by [Rdjarbeng](#)

Start- 27 June, 2023 Last update- 22nd June, 2023 Have feedback or comments? Open an issue or contact me on [LinkedIn](#) or social media if you want to suggest improvements.